

What is claimed is:

1. A processing device to support parallel execution of multiple instructions, the
5 processing device comprising:
a trace detector that identifies traces in a segment of code including
successive instructions, each of multiple identified traces in the segment of code
including a set of instructions capable of being executed on an execution unit; and
a dependency detector that, prior to parallel execution of multiple
10 identified traces on corresponding execution units, analyzes the traces identified
in the segment of code to determine a dependency order for executing the traces,
the dependency order identifying at least one of the traces associated with the
segment of code that cannot be properly executed in parallel with another trace in
the segment of code.
15
2. A processing device as in claim 1, wherein the traces each include a sequence of
contiguous instructions intended to be executed successively in time and the
dependency order indicates which of the multiple traces must be executed before
others identified in the segment of code.
20
3. A processing device as in claim 2 further comprising:
a scheduler that schedules parallel execution of traces detected within a
basic block of JAVA code on multiple execution units according to the
dependency order.
25
4. A processing device as in claim 1, wherein the trace detector identifying traces in
the segment of code includes identifies operand stack dependencies associated
with portions of the segment of code and wherein the corresponding execution
units each include an operand stack.
30

5. A processing device as in claim 1, wherein the dependency detector analyzes the traces to determine data dependencies associated with traces in the segment of code and identify the dependency order for executing at least some of the traces in parallel at run time.

5

6. A processing device as in claim 1 further comprising:
multiple execution units to execute multiple traces in parallel based on the dependency order;

10

a buffer to temporarily store results associated with execution of multiple executed traces; and

a comparator circuit that, at run time of executing the multiple traces in parallel, identifies an out-of-order memory dependency condition associated with parallel executed traces resulting in an error; and

15

the comparator circuit, in response to identifying the out-of-order memory dependency condition:

squashes execution of latter traces in the segment of code that depend on results from earlier traces;

clears results in the temporary buffer associated with the squashed traces; and

20

reschedules squashed traces for later execution.

7. A processing device as in claim 1 further comprising:
multiple execution units that execute multiple traces in parallel based on the dependency order;

25

a temporary buffer to store results associated with execution of multiple traces of the segment of code; and

30

a comparator circuit to identify that an out-of-order memory dependency condition associated with parallel executed traces did not occur at run time of executing the multiple traces in parallel, the comparator circuit, in response to identifying that an out-of-order memory dependency condition did not occur at

run time of a particular trace, loads the results stored in the temporary buffer to memory after the particular trace completes execution.

8. A processing device as in claim 1, wherein at least one of the traces is processed to include a folded bytecode instruction replacing a corresponding sequence of
5 bytecode instructions.

9. A processing device as in claim 1 further comprising:
a fetcher that fetches multiple code instructions from different traces
10 identified in the segment of code;
a decoder that decodes the multiple fetched code instructions into
corresponding bytecode instructions; and
a buffer unit to store the bytecode instructions associated with the multiple
decoded code instructions in corresponding trace buffers for each trace.

15
10. A processing device as in claim 1 further comprising:
a temporary buffer that stores results associated with execution of multiple
traces; and
wherein the comparator circuit identifies an out of order memory
20 dependency condition based on a search for:
i) a READ after a WRITE to the same memory address for different
parallel executed traces,
ii) a WRITE after a READ to the same memory address for different
parallel executed traces, and
25 iii) a WRITE after a WRITE to the same memory address for different
parallel executed traces.

11. A processing device as in claim 1, wherein the dependency detector analyzing the
traces in the segment of code determines a dependency order by comparing
30 memory access instructions in a first trace to memory access instructions in other
traces identified in the segment of code to identify a potential trace dependency in

which the first trace contains a memory access instruction that depends on the operation of another memory access instruction in at least one of the other traces identified in the segment of code.

- 5 12. A processing device as in claim 1, wherein the trace detector identifying traces within the segment of code:

 identifies a beginning trace instruction in the segment of code whose operation corresponds to a first clean condition of an execution unit;

10 identifies a subsequent trace instruction in the segment of code whose operation corresponds to a non-clean condition of the execution unit; and

 identifies an ending trace instruction in the segment of code whose operation follows the first clean condition and the non-clean condition of the execution unit and that corresponds to at least one of:

15 i) a second clean condition of the execution unit; and

 ii) an end of the segment of code; and

 designates, as a trace within the segment of code, all instructions in the segment of code including, and in-between, the beginning trace instruction and the ending trace instruction.

- 20 13. A processing device as in claim 1, wherein the dependency detector identifying the dependency order, upon completion of execution of at least two traces, updates the dependency order to remove any trace dependencies associated with other non-executed traces that depended on completion of execution of the at least two executed traces.

25

14. A method associated with parallel execution of multiple instructions, the method comprising:

30 identifying traces in a segment of code including successive instructions, each of multiple identified traces in the segment of code including a set of instructions capable of being executed on an execution unit; and

prior to parallel execution of multiple identified traces on corresponding execution units, analyzing the traces identified in the segment of code to determine a dependency order for executing the traces, the dependency order identifying at least one of the traces associated with the segment of code that cannot be properly executed in parallel with another trace in the segment of code.

15. A method as in claim 14, wherein the traces each include a sequence of contiguous instructions intended to be executed successively in time and the dependency order indicates which of the multiple traces must be executed before others identified in the segment of code.

16. A method as in claim 15 further comprising:
scheduling parallel execution of traces detected within a basic block of JAVA code on multiple execution units according to the dependency order.

17. A method as in claim 14, wherein identifying traces in the segment of code includes identifying operand stack dependencies associated with portions of the segment of code and wherein the corresponding execution units each including an operand stack.

18. A method as in claim 14, wherein analyzing the traces includes determining data dependencies associated with traces in the segment of code to identify the dependency order for executing at least some of the traces in parallel at run time.

19. A method as in claim 14 further comprising:
executing multiple traces in parallel based on the dependency order;
temporarily storing results associated with execution of the multiple traces in a temporary buffer; and
at run time, identifying an out-of-order memory dependency condition associated with parallel executed traces resulting in an error; and
in response to identifying the out-of-order memory dependency condition:

squashing execution of latter traces in the segment of code that depend on results from earlier traces and clearing results in the temporary buffer associated with the squashed traces; and
rescheduling squashed traces for later execution.

5

20. A method as in claim 14 further comprising:
executing multiple traces in parallel based on the dependency order;
storing results associated with execution of multiple traces of the segment of code in a temporary buffer; and
10 identifying that an out-of-order memory dependency condition associated with parallel executed traces did not occur at run time of executing the multiple traces in parallel; and
in response to identifying that an out-of-order memory dependency condition did not occur at run time of a particular trace:
15 storing the results stored in the temporary buffer to memory after the trace completes execution.

21. A method as in claim 14, wherein at least one of the traces is processed to include a folded JAVA bytecode instruction replacing a corresponding sequence of JAVA
20 bytecode instructions.

22. A method as in claim 14 further comprising:
fetching multiple code instructions from different traces identified in the segment of code;
25 decoding the multiple fetched code instructions into corresponding bytecode instructions; and
storing the bytecode instructions associated with the multiple decoded code instructions in corresponding trace buffers for each trace.

- 30 23. A method as in claim 14 further comprising:

storing results associated with execution of multiple traces in a temporary buffer; and

wherein identifying an out of order memory dependency condition includes searching for:

- 5 i) a READ after a WRITE to the same memory address for different parallel executed traces,
- ii) a WRITE after a READ to the same memory address for different parallel executed traces, and
- iii) a WRITE after a WRITE to the same memory address for different
10 parallel executed traces.

24. A method as in claim 14, wherein analyzing the traces in the segment of code to determine a dependency order includes:

 comparing memory access instructions in a first trace to memory access
15 instructions in other traces identified in the segment of code to identify a potential trace dependency in which the first trace contains a memory access instruction that depends on the operation of another memory access instruction in at least one of the other traces identified in the segment of code.

20 25. A method as in claim 14, wherein identifying traces within the segment of code comprises:

 identifying a beginning trace instruction in the segment of code whose operation corresponds to a first clean condition of an execution unit;

 identifying a subsequent trace instruction in the segment of code whose
25 operation corresponds to a non-clean condition of the execution unit; and

 identifying an ending trace instruction in the segment of code whose operation follows the first clean condition and the non-clean condition of the execution unit and that corresponds to at least one of:

- i) a second clean condition of the execution unit; and
30 ii) an end of the segment of code; and

designating, as a trace within the segment of code, all instructions in the segment of code including, and in-between, the beginning trace instruction and the ending trace instruction.

- 5 26. A method as in claim 14, wherein identifying the dependency order further comprises:

 upon completion of execution of at least two traces, updating the dependency order to remove any trace dependencies associated with other non-executed traces that depended on completion of execution of the at least two
10 executed traces.

27. A computer program product including a computer-readable medium having instructions stored thereon for processing data information, such that the instructions, when carried out by a processing device, enable the processing
15 device to perform the steps of:

 identifying traces in a segment of code including successive instructions, each of multiple identified traces in the segment of code including a set of instructions capable of being executed on an execution unit; and

 prior to parallel execution of multiple identified traces on corresponding
20 execution units, analyzing the traces identified in the segment of code to determine a dependency order for executing the traces, the dependency order identifying at least one of the traces associated with the segment of code that cannot be properly executed in parallel with another trace in the segment of code.

- 25 28. A processing device to support parallel execution of multiple instructions, the processing device comprising:

 means for identifying traces in a segment of code including successive instructions, each of multiple identified traces in the segment of code including a set of instructions capable of being executed on an execution unit; and

30 means for analyzing the multiple identified traces identified in the segment of code, prior to parallel execution of the multiple identified traces on

corresponding execution units, to determine a dependency order for executing the multiple identified traces, the dependency order identifying at least one of the traces associated with the segment of code that cannot be properly executed in parallel with another trace in the segment of code.

5

29. A processing device to support parallel execution of multiple instructions, the processing device comprising:

a fetcher to fetch instructions;

10

a trace detector coupled to receive the fetched instructions, the trace detector identifying traces in a segment of code including successive instructions, each of multiple identified traces in the segment of code including a set of instructions capable of being executed on an execution unit;

15

a dependency detector that, prior to parallel execution of multiple identified traces on corresponding execution units, analyzes the traces identified in the segment of code to determine a dependency order for executing the traces, the dependency order identifying at least one of the traces associated with the segment of code that cannot be properly executed in parallel with another trace in the segment of code;

20

a trace scheduler coupled to the dependency detector and the trace detector, the trace scheduler receiving a set of traces and, based on the dependency order, causing the corresponding execution units to execute traces within the set of traces in parallel, the execution taking place in an execution order that is based on the identified dependency order, at least two traces being executed in parallel and if the dependency order indicates that a second trace is dependent upon a first trace, the first trace being executed prior to the second trace; and

25

multiple execution units to execute the traces in parallel.

30. A processing device as in claim 29 further comprising:

30

a temporary buffer coupled to the execution units to store results associated with execution of multiple traces of the segment of code; and

a comparator circuit to detect whether an out-of-order memory dependency condition associated with parallel executed traces occurs at run-time of executing the multiple traces in parallel, the comparator circuit conditionally loading the results stored in the temporary buffer to memory after the particular trace completes execution.

31. A processing device as in claim 30 further comprising a squash circuit coupled to receive a signal from the comparator circuit identifying detection of an out-of-order memory dependency condition, the squash circuit:

squashing execution of latter traces in the segment of code that depend on results from earlier traces;
clearing results in the temporary buffer associated with the squashed traces; and
generates a signal to the trace scheduler to reschedule squashed traces for later execution.

32. A processing device as in claim 29 further comprising:

a basic block trace table cache to store trace information associated with a currently executed method.

33. A processing device as in claim 32 further comprising:

bytecode trace fetch logic that utilizes multiple program counters stored in the basic block trace table cache to order the fetcher to fetch multiple instructions from multiple locations of a method cache.

34. A processing device as in claim 32 further comprising:

a decoded bytecode trace buffer including individual buffers, each individual buffer storing instructions for a given trace.

35. A processing device as in claim 34, wherein the trace scheduler:

identifies non-dependent traces based on the trace information in the basic block trace table cache;

selects the set of traces to be executed on corresponding execution units;

5 allocates execution units to execute the set of traces in parallel; and
 fetches the set of traces from the decoded bytecode trace buffer for parallel execution by the execution units.

10 36. A processing device as in claim 29, wherein the execution units each includes an operand stack, a reservation station and an associated functional unit.

 37. A processing device as in claim 29, wherein the execution units each includes multiple sets of shared local variable registers, each set of local variable registers being utilized by a corresponding method.

15

 38. A processing device as in claim 29, wherein the execution units each includes:
 a load buffer and a store buffer to temporarily store retrieved and modified data associated with multiple parallel executed traces in a scratchpad area.

20

25